REMARKS

Claims 1-2, 24, 25, 32, and 33 have been amended. Attached hereto is a marked-up version of the changes made to the specification and claims by the current amendment. The attached page is captioned "Version with markings to show changes made." Claims 8-23, 26-31, and 34-42 have been cancelled as corresponding to non-elected claims. Consequently, claims 1-7, 24-25, and 32-33 are pending.

Claim 1 stands rejected under 35 U.S.C. § 102(b) as being anticipated by Record (U.S. Patent No. 5,335,383). Claims 2, 3, 24, and 32 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Record in view of Lehr (U.S. Patent No. 5,898,873). Claim 4 stands rejected under 35 U.S.C. § 103(a) as being unpatentable over Record, Lehr, and U.S. Patent No. 6,212,544 (Borkenhagen). Claim 5-6 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Record, Lehr, and Ellis. Claims 7, 25, and 33 stand rejected under 35 U.S.C. § 103(a) over Record and Toutonghi (U.S. Patent No. 5,842,016). These rejections are respectfully traversed.

The present invention is directed to an apparatus and method for granting and revoking exclusive access rights to one or more threads executing on a processor based system. A thread is a basic unit of execution in a computer system. Processes are groups of at least one thread. In a multi-tasking computer system, a scheduler component of the operating system is used to context switch the execution state of the computer system among the plurality of threads. In the prior art, exclusive access is granted to a thread through the use of traditional mutual exclusion mechanisms, such as semaphores.

The present invention discloses an apparatus and method for granting exclusive access to a resource by controlling thread scheduling. A scheduler in accordance with the principles of the present invention monitors the state of a scheduling flag, which is typically in a "second" state. When the scheduling flag is in the second state, the scheduler performs context switching of the processor's execution state among the plurality of threads on the computer system in accordance to a scheduling policy. However, if the scheduling flag is in a "first" state, the scheduler does not perform context switching as

long as the scheduling flag is in the first state. An application programming interface (API) is provided to permit threads to call functions which respectively reads, writes a first state, or writes a second state to the scheduling flag. Thus, in accordance to one aspect of the present invention, a thread may gain exclusive access over one or more resources on a computer system by calling an API routine which sets the scheduling flag to the first state, thereby ensuring that no other thread will manipulate any resources until the second API command is issued. See, e.g., specification at pp. 11-12. Accordingly, independent claims 1, 24, and 32 recite:

> first means responsive to a predetermined first application program interface call from one of said plurality of threads for setting a prescribed flag to one of first and second states; second means for detecting, after said flag is set to said one state, a prescribed change in a state of said scheduling policy, and for setting said flag to a other one of said first and second states (claim 1)
>
> in response to an application program interface call ... setting a flag indicating absence of context switching; disabling context switching between threads when said flag is set to correspond to absence of context switching; ... setting said flag to a state corresponding to presence of context switching; enabling context switching between threads when said flag is set to correspond to presence of context switching (claim 24)
>
> in response to an application program interface call from a thread which requests start of detection of a presence or absence of context switching, setting a flag indicating the absence of context switching; in response to an application program interface call from the thread, and after said flag is set to the state corresponding to the absence of context switching, setting said flag to a state corresponding to presence of context switching; (claim 32)

According to another aspect of the present invention, the API functions may include a fourth API routine which is used to establish a flag taking a state which reflects whether a region of memory has been written, and a fifth API routine which reads the value of the flag. In this manner, one thread can monitor whether another thread has written the memory address range of interest. If the memory address range is carefully chosen, the ability to known that the memory address

range was not written can be used to ensure that access to a resource was exclusively to a thread, while the ability to known that the memory address was written to could be used to take corrective action. See specification, p. 13. Accordingly, independent claims 25 and 33 recite:

> in response to an application program interface call from a thread from a plurality of threads which requests start of detection of a presence or absence of a data write to a designated memory area, setting a flag indicating the absence of a data write; setting said flag to a state corresponding to presence of a data write when there is a data write to said designated memory area; and in response to an application program interface call from said thread which interface request termination of detection of presence/absence of a data write to the designated memory area (claim 25)
>
> in response to an application program interface call from a thread which requests start of detection of a presence or absence of a data write to a designated memory area, setting a flag indicating an absence of data writes; setting said flag to a state corresponding to presence of a data write when there is a data write to said designated memory area; and in response to an application program interface call from said thread which interface requests termination of detection of presence/absence of a data write to the designated memory area, returning a value corresponding to the state of aid flag to said thread. (claim 33)

Record is directed to an event management system for a computer operating system. More specifically, Record teaches a general purpose event management system which includes an event manager (e.g., Fig. 1, manager 26) which interacts with event definitions (e.g., Fig. 1, definitions 16a-16d), event monitors (e.g., Fig 1, monitors 28a-28d), and event handlers (e.g., Fig. 1, handlers 14a-14d). Application programs executing on a computer system with the event management system of Record may define a variety of events (e.g., completion of a function, reaching specific checkpoints in computer code, application error exceptions), and have the event management system call an appropriate event handler if an event occurs. See column 6, lines 4-67; Figs. 5-6. Record therefore teaches a flexible event management system. However, the claims as amended, either

require (1) disabling/enabling context switching based on the state of a flag which can be controlled via API routines; or (2) monitoring setting up a flag and having the flag determine whether a designated memory region has been written. None of the limitations, as reproduced above with respect to independent claims 1, 24, 32, 25, and 33, are taught or suggested by Record.

Lehr is directed to a computer execution trace analysis tool. Computer execution traces are logs which report the execution activity of a computer system. Programmers may need to use execution traces while debugging to analyze exactly what a piece of computer code is doing. One issue which arises when interpreting traces is that in multi-tasking operating systems, one process or thread may be context switched for another process of thread, in accordance with the policy of a scheduler component of the operating system. In some computer systems, context switches are easily detected, while in others, especially operation systems which depend upon "cooperative multitasking" (e.g., Microsoft Windows), context switches are less predictable because a context switch can only occur when, for example, a Windows program calls an API routine to examine the event queue. Although Lehr understands the concept of a context switch, Lehr fails disclose or suggest an ability to enable or disable context switches by a scheduler, and controlling the scheduler using a flag whose state can be changed by calling an API routine.

Toutonghi discloses a garbage collection system in a computer system supporting threads. Toutonghi discloses a system which does not actually perform garbage collection (even after garbage collection has been "initiated") until each thread exits each resource accessing section. This minimizes the amount of time each thread is negatively impacted by the garbage collection process. See column 2, lines 32-63. Toutonghi discloses, *inter alia,* two API routines: DisableGarbageCollection and EnableGarbageCollection and these two routines respectively set and reset a DisableGC flag. However, contrary to the assertion made in the Office Action, Toutonghi does not disclose or suggest "setting said flag to a state corresponding to presence of a data write when there is data write to the designated memory area." The EnableGarbageCollection API merely resets the DisableGC flag and the state of the DisableGC flag does not indicate

whether a memory range has been written (although the garbage collection process may write that region after the DisableGC flag has been reset). In the above recited limitations of claims 25 and 33, the flag is written when a write to the relevant memory address range is performed.

The Office Action additionally cites to Borkenhagen and Ellis for various teachings which are allegedly relevant to some of the depending claims. However, neither Borkenhagen and Ellis, whether taken singly or in combination, discloses or suggests the above-recited limitations of the independent claims. Accordingly, independent claims 1, 24, 25, 32, and 33 are believed to be allowable over the prior art of record. Claims 2-7, which depend from claim 1, are also believed to be allowable over the prior art of record for these reasons and because the combinations defined in the claims are not shown or suggested by the cited references.

In view of the above, each of the presently pending claims in this application is believed to be in immediate condition for allowance. Accordingly, the Examiner is respectfully requested to withdraw the outstanding rejection of the claims and to pass this application to issue.

Dated: March 19, 2003

Respectfully submitted,

By_____

Thomas J. D'Amico
Registration No.: 28,371
Christopher S. Chow
Registration No.: 46,493
DICKSTEIN SHAPIRO MORIN & OSHINSKY LLP
2101 L Street NW
Washington, DC 20037-1526
(202) 785-9700
Attorneys for Applicant

Version With Markings to Show Changes Made

Please amend claims 1-2, 24-25, and 32-33 as follows:

1. A program control apparatus for controlling execution of a program in a computer system in which a plurality of threads [is] are switched in accordance with a scheduling policy by a scheduler, comprising:

first means responsive to a predetermined first application program interface call from [a] one of said plurality of threads for setting a prescribed flag to one of first and second states;

second means for detecting, after said flag is set to said one state, a prescribed change in a state of said scheduling policy, [computer system,] and for setting said flag to a [the] other one of said first and second states; and

third means responsive to a predetermined second application program interface call constituting a pair with said first application program interface, from said thread, for returning a value indicative of the state of said flag to said thread.

2. The program control apparatus according to claim 1, wherein

said first means includes means responsive to an application program interface call from [a] one of said plurality of threads which [interface] requests start of detection of presence/absence of a context switching, for setting a flag indicating presence/absence of a context switching to a state indicating absence of a context switching;

said second means includes means for setting, after said flag is set to the state corresponding to the absence of a context switching, [and a scheduler switches a context,] said flag to a state corresponding to presence of a context switching; and

said third means includes means responsive to an application program interface call from one of said plurality of threads which [interface] requests termination of detection of

presence/absence of a context switching, for returning a value corresponding to the state of said flag to said thread.

24. A method of program control, comprising the steps of:

in response to an application program interface call from a thread from a plurality of threads which [interface] requests start of detection of a presence or absence [presence/absence] of [a] context switching, setting a flag indicating absence of [presence/absence of a] context switching; [to a state corresponding to absence of context switching;]

disabling context switching between threads when said flag is set to correspond to absence of context switching;

after said flag is set to the state corresponding to [the] absence of [a] context switching, [when a context switch is switched by a scheduler,] setting said flag to a state corresponding to presence of [a] context switching; [and]

enabling context switching between threads when said flag is set to correspond to presence of context switching; and

in response to an application program interface call from said thread which interface requests termination of detection of such thread.

25. A method of program control, comprising the steps of:

in response to an application program interface call from a thread from a plurality of threads which [interface] requests start of detection of a presence or absence

[presence/absence] of a data write to a designated memory area, setting a flag indicating the absence [presence/absence] of a data write; [to a state corresponding to absence of the data write;]

setting said flag to a state corresponding to presence of a data write when there is a data write to said designated memory area; and

in response to an application program interface call from said thread which interfaced requests termination of detection of presence/absence of a data write to the designed memory area, returning a value corresponding to the state of said flag to said thread.

32. A computer readable recording medium storing a program control program allowing a computer to execute a program control method, said program control method includes the steps of

in response to an application program interface call from a thread which [interface] requests start of detection of a presence or absence [presence/absence] of [a] context switching, setting a flag indicating the [a] absence [presence/absence] of [a] context switching; [to a state corresponding to absence of a context switching;]

in response to an application program interface call from the thread, and after said flag is set to the state corresponding to the absence of [a] context switching, [when a context is switched by a scheduler,] setting said flag to a state corresponding to presence of [a] context switching; and

in response to an application program interface call from said thread which [interface] requests termination of detection of the presence or absence of [presence/absence of] a context switching, returning a value corresponding to the state of said flag to said thread.

33. A computer readable recording medium storing a program control program allowing a computer to execute a program control method, said program control method comprising [including] the steps of:

in response to an application program interface call from a thread which [interface] requests start of detection of a presence or absence [presence/absence] of a data write to a designated memory area, setting a flag indicating an absence of data writes; [presence/absence of a data write to a state corresponding to an absence of a data write;]

setting said flag to a state corresponding to presence of a data write when there is a data write to said designated memory area; and

in response to an application program interface call from said thread which interface requests termination of detection of presence/absence of a data write to the designated memory area, returning a value corresponding to the state of aid flag to said thread.